CDC

DesignWare minPower Building Block

DW_gray_sync

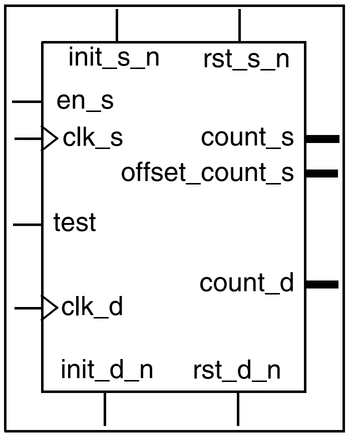
Gray Coded Synchronizer

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

Revision History

- Interface between dual asynchronous clock domains
- Parameterized counter width
- Parameterized number of synchronizing stages
- Parameterized test feature
- Outputs to source can be configured as registered or not
- Output to destination domain can be configured as registered or not
- Model missampling of data on source clock domain
- Parameterized pipelining source domain results to destination domain
- Includes a low-power implementation that has power benefits from minPower optimization (for details, see Table 1-3 on page 3)



Description

The DW_gray_sync includes a binary counter in the source domain whose value is recoded as a binary-reflected-Gray code. The recoded count value is synchronized to the destination domain and then decoded from binary-reflected-Gray back to binary. The binary count value and an offset version (which is only useful for non integer power of two sequences) are both available in the `clk_s` domain as `count_s` and `offset_count_s`. This synchronizer is used within other synchronizers to pass incrementing pointers across domains.

A unique built-in verification feature allows the designer to turn on a random sampling error mechanism that models skew between bits of the internal binary-reflected-Gray coded counter bus derived from the source domain (for more details, see "Simulation Methodology" on page 6). This facility provides an opportunity for determining system robustness during the early development phases and without having to develop special test stimulus.

Table 1-1 Pin Descriptions

Pin Name	Width	Direction	Function
clk_s	1	Input	Source domain clock source
rst_s_n	1	Input	Source domain asynchronous reset (active low)
init_s_n	1	Input	Source domain synchronous reset (active low)
en_s	1	Input	Source domain counter advance initiation

Table 1-1 Pin Descriptions (Continued)

Pin Name	Width	Direction	Function
count_s	width	Output	Source domain counter value
offset_count_s	width	Output	Source domain offset counter value
clk_d	1	Input	Destination domain clock source
rst_d_n	1	Input	Destination domain asynchronous reset (active low)
init_d_n	1	Input	Destination domain synchronous reset (active low)
count_d	width	Output	Destination domain counter value (synchronized version of `count_s`)
test	1	Input	Scan test mode select

Table 1-2 Parameter Description

Parameter	Values	Description
width	1 to 1024 Default: 8	Vector width of input `data_s` and output `data_d`
offset	0 to $(2^{(width-1)}) - 1$	Offset for non integer power of 2 counter value Default: 0
reg_count_d	0 or 1 Default: 1	Register `count_d` output ■ 0: `count_d` not registered ■ 1: `count_d` registered
f_sync_type	0 to 4 Default: 2	Forward synchronization type Defines type and number of synchronizing stages: ■ 0: Single clock design, no synchronizing stages implemented ■ 1: 2-stage synchronization with first stage negative-edge capturing and second stage positive-edge capturing ■ 2: 2-stage synchronization with both stages positive-edge capturing ■ 3: 3-stage synchronization with all stages positive-edge capturing ■ 4: 4-stage synchronization with all stages positive-edge capturing
tst_mode	0 to 2 Default: 0	Test mode ■ 0: No latch is inserted for scan testing ■ 1: Insert negative-edge capturing flip-flip on `data_s` input vector when `test` input is asserted ■ 2: Insert hold latch using active-low latch

Table 1-2 Parameter Description (Continued)

Parameter	Values	Description
verif_en	0 to 4 Default: 2	Verification enable control ■ 0: No sampling errors inserted ■ 1: Sampling errors are randomly inserted with 0 or up to 1 destination clock cycle delays ■ 2: Sampling errors are randomly inserted with 0, 0.5, 1, or 1.5 destination clock cycle delays ■ 3: Sampling errors are randomly inserted with 0, 1, 2, or 3 destination clock cycle delays ■ 4: Sampling errors are randomly inserted with 0 or up to 0.5 destination clock cycle delays For more information about *verif_en*, see "Simulation Methodology" on page 6.
pipe_delay	0 to 2 Default: 0	Pipeline Binary to Gray Code results ■ 0: No pipelining other than re-timing register ■ 1: One additional pipeline stage ■ 2: Two additional pipeline stages
reg_count_s	0 to 1 Default: 1	Register `count_s` output ■ 0: `count_s` not registered ■ 1: `count_s` registered
reg_offset_ count_s	0 to 1 Default: 1	Register `offset_count_s` output ■ 0: `offset_count_s` not registered ■ 1: `offset_count_s` registered

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare
lpwr[a]	Low Power synthesis model	■ DesignWare (P-2019.03 and later) ■ DesignWare-LP (before P-2019.03)

a. Requires that you enable minPower; for details, see "Enabling minPower" on page 9.
When minPower is enabled, the lpwr implementation is always chosen during synthesis.

Table 1-4 Simulation Models

Model	Function
DW03.DW_GRAY_SYNC _CFG_SIM	Design unit name for VHDL simulation

Table 1-4 Simulation Models (Continued)

Model	Function
dw/dw03/src/DW_gray_sync _sim.vhd	VHDL simulation model source code (modeling RTL) - no missampling
dw/dw03/src/DW_gray_sync _sim_ms.vhd	VHDL simulation model source code (modeling RTL) - missampling
dw/sim_ver/DW_gray_sync.v	Verilog simulation model source code

Block Diagram

Figure 1-1 DW_gray_sync Basic Block Diagram

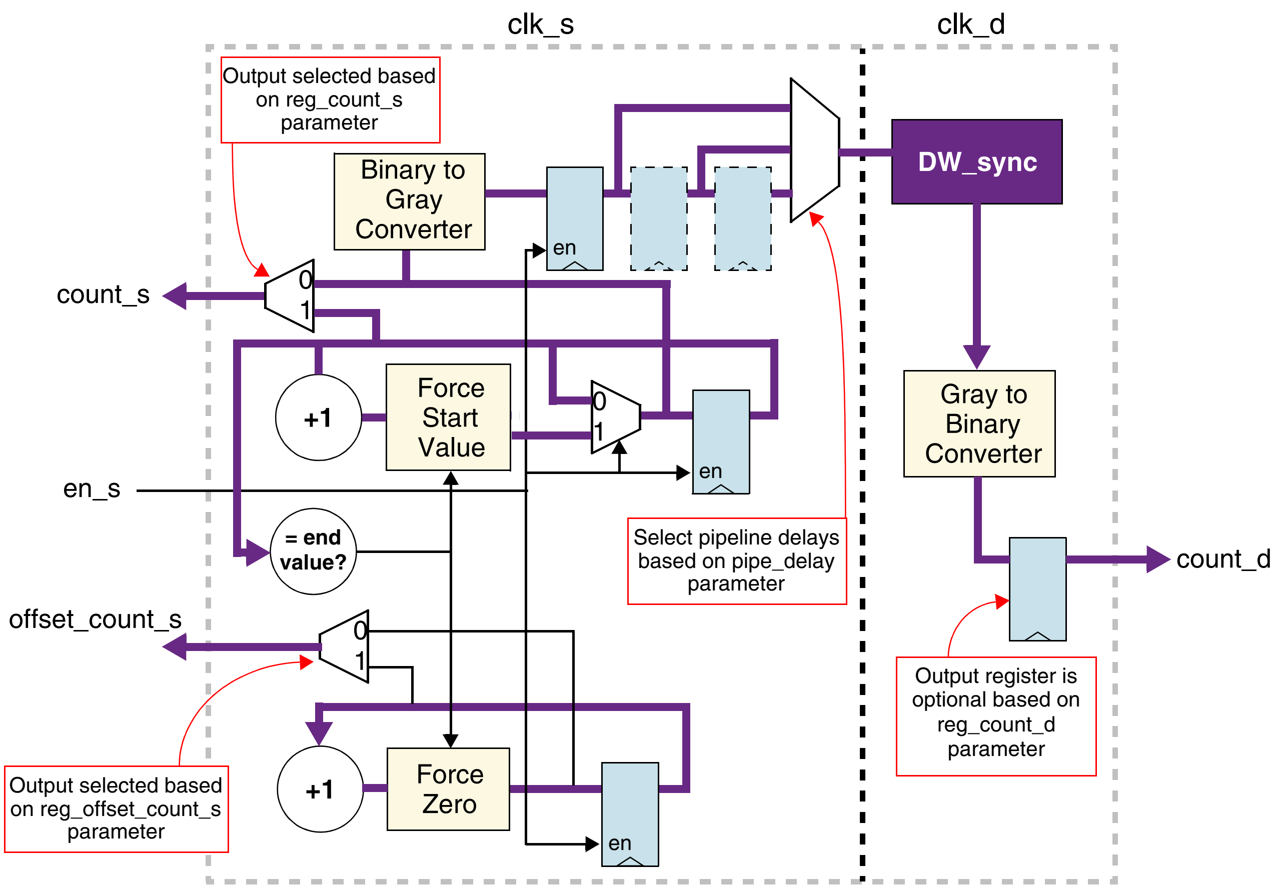

clk_s
clk_d
Output selected based on reg_count_s parameter
Binary to Gray Converter
DW_sync
en
count_s
0
1
Force Start Value
+1
en_s
= end value?
Select pipeline delays based on pipe_delay parameter
Gray to Binary Converter
count_d
offset_count_s
+1
Force Zero
Output register is optional based on reg_count_d parameter
Output selected based on reg_offset_count_s parameter

Table 1-5 Reflected Gray Code Sequencing Example

Reflected Binary Gray Code Sequencing for a 4-bit Domain							
Binary Sequence	**Full Gray Sequence**		**14-State Gray Sequence**	**Binary Offset Sequence**		**10-State Gray Sequence**	**Binary Offset Sequence**
0000	0000 (start)		—	—		—	—
0001	0001		0001 (start)	0000		—	—
0010	0011		0011	0001		—	—
0011	0010		0010	0010		0010 (start)	0000
0100	0110		0110	0011		0110	0001
0101	0111		0111	0100		0111	0010
0110	0101		0101	0101		0101	0011
0111	0100		0100	0110		0100	0100
Axis of Symmetry							
1000	1100		1100	0111		1100	0101
1001	1101		1101	1000		1101	0110
1010	1111		1111	1001		1111	0111
1011	1110		1110	1010		1110	1000
1100	1010		1010	1011		1010 (end)	1001
1101	1011		1011	1100		—	—
1110	1001		1001 (end)	1101		—	—
1111	1000 (end)		—	—		—	—

Reset Considerations

It is recommended that both domains be reset at or around the same time. If the source domain interface is reset and the destination domain is not, the destination domain `clk_d` value will be invalid relative to the source domain `clk_s` for at least *f_sync_type* number of destination domain clock cycles depending on the *reg_count_d* setting. If *reg_count_d* is 1, the total number of destination clock cycles before `clk_d` reflects the value on `clk_s` after the source domain reset is *f_sync_type* + 1.

Simulation Methodology

Because this component contains synchronizing devices, there are two methods available for simulation. One method is to use the simulation models that emulate the RTL model. Or, you can enable modeling of random skew between bits of signals traversing to and from each domain (called missampling).

To use the simulation models that emulate the RTL model, no special configuration is required.

To use missampling requires the following considerations:

- To enable missampling in Verilog simulations, define the macro DW_MODEL_MISSAMPLES:

  ```
  `define DW_MODEL_MISSAMPLES
  ```

  If `DW_MODEL_MISSAMPLES is defined, the *verif_en* parameter comes into play to configure the simulation model as described by Table 1-2 on page 2. If `DW_MODEL_MISSAMPLES is not defined, the Verilog simulation model behaves as if *verif_en* is set to 0.

- To enable missampling in VHDL simulations, a simulation architecture named sim_ms is provided. The parameter *verif_en* only has meaning when using sim_ms. That is, when the `sim` simulation architecture is used instead, the model behaves as though *verif_en* is set to 0. For examples of how each architecture is used, see "HDL Usage Through Component Instantiation - VHDL" on page 10.

Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

  ```
  WARNING: <instance_path>.<clock_name>_monitor:
      at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
  ```

  To suppress only this warning message for all DWBB components, use the following macro:

  - ❑ Define the DW_DISABLE_CLK_MONITOR macro. You can define this macro in the following ways:
    - Specify the Verilog preprocessing macro in Verilog code:

      ```
      `define DW_DISABLE_CLK_MONITOR
      ```

    - Or, include a command line option to the simulator, such as:

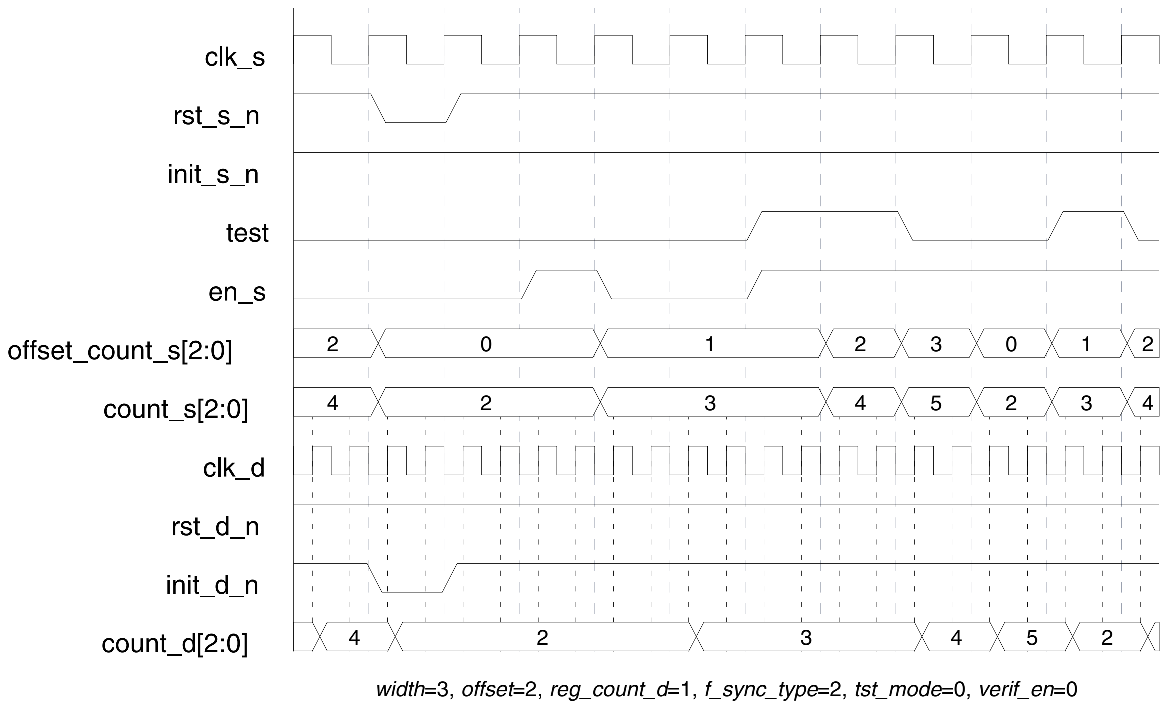
      `+define+DW_DISABLE_CLK_MONITOR` (which is used for the Synopsys VCS simulator)

  This message is also suppressed using the DW_SUPPRESS_WARN macro explained earlier.

Timing Diagrams

Figure 1-2 depicts the fundamental operation of advancing the counters after resetting both clock domains. Notice that the *offset* parameter is set to 2, which sets the `count_s` initial value upon reset to 2 and advances to a maximum value of 5. The `count_d` result is three `clk_d` cycles after the change in `count_s`, because *f_sync_type* is set to 2 and *reg_count_d* is set to 1. The other parameter settings are: *width* = 3, *offset* = 2, *tst_mode* = 0, *verif_en* = 0, *pipe_delay* = 0, *reg_count_s* = 1, and *reg_offset_count_s* = 1.

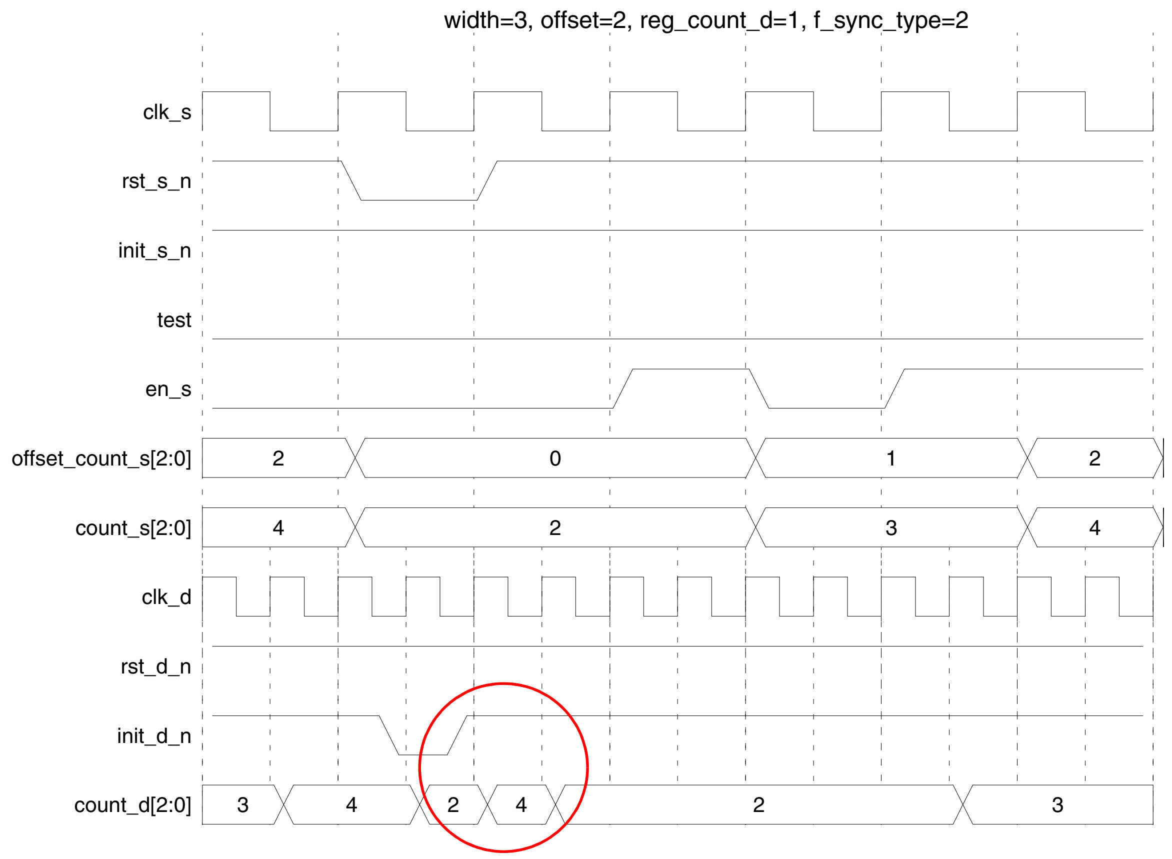
Figure 1-2 Advancing Counters Including Reset of Both Domains



Beginning in Release F-2011.09, the underlying DW_sync component in DW_gray_sync does not connect `init_d_n`. Instead, DW_sync has `init_d_n` de-asserted. As a by-product of this, the resetting of `count_d` is based on the cleared state of the gray code entering and propagating through DW_sync. Figure 1-3 on page 8 shows an example of this. In this case, the same scenario is presented as in Figure 1-2, but here, `init_d_n` is only a single-cycle of `clk_d`. Here it can been seen that the `count_d[2:0]` goes to its reset state of '2' after `init_d_n` is asserted, but after `init_d_n` is de-asserted, `count_d[2:0]` goes to '4' for one `clk_d` cycle before going back to '2' on the next cycle (see area circled in red). This exhibits the fact that the reset value going into the DW_sync has to make its way through DW_sync and into the destination domain, and `init_d_n` was not held asserted long enough to mask the non-reset value. In Figure 1-2, with `init_d_n` being asserted two cycles of `clk_d`, it was long enough to prevent sampling of the non-reset value exiting DW_sync and, hence, `count_d[2:0]` remained at '2'.

This behavior, in fact, would also be seen should the source domain not get reset prior to the destination domain reset (for more information, see "Reset Considerations" on page 5).

Figure 1-3 Invalid 'count_d' Result During Reset



Enabling minPower

You can instantiate this component without enabling minPower, but to achieve power savings from the low-power implementation "lpwr" (see Table 1-3 on page 3), you must enable minPower optimization, as follows:

- Design Compiler
  - Version P-2019.03 and later:

    ```
    set power_enable_minpower true
    ```
  - Before version P-2019.03 (requires the DesignWare-LP license feature):

    ```
    set synthetic_library {dw_foundation.sldb dw_minpower.sldb}
    set link_library {* $target_library $synthetic_library}
    ```
- Fusion Compiler

  Optimization for minPower is enabled as part of the total_power metric setting. To enable the total_power metric, use the following:

  ```
  set_qor_strategy -stage synthesis -metric total_power
  ```

Related Topics

- Memory – Registers Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp.all;

entity DW_gray_sync_inst is
      generic (
        inst_width : INTEGER := 8;
        inst_offset : INTEGER := 0;
        inst_reg_count_d : INTEGER := 1;
        inst_f_sync_type : INTEGER := 2;
        inst_tst_mode : INTEGER := 0;
        inst_verif_en : INTEGER := 2;
            inst_pipe_delay : NATURAL:= 0;
            inst_reg_count_s : NATURAL := 1;
            inst_reg_offset_count_s : NATURAL := 1
        );
      port (
        inst_clk_s : in std_logic;
        inst_rst_s_n : in std_logic;
        inst_init_s_n : in std_logic;
        inst_en_s : in std_logic;
        count_s_inst : out std_logic_vector(inst_width-1 downto 0);
        offset_count_s_inst : out std_logic_vector(inst_width-1 downto 0);

        inst_clk_d : in std_logic;
        inst_rst_d_n : in std_logic;
        inst_init_d_n : in std_logic;
        count_d_inst : out std_logic_vector(inst_width-1 downto 0);

        inst_test : in std_logic
        );
    end DW_gray_sync_inst;


architecture inst of DW_gray_sync_inst is
begin

    -- Instance of DW_gray_sync
    U1 : DW_gray_sync
    generic map ( width => inst_width, offset => inst_offset,
            reg_count_d => inst_reg_count_d, f_sync_type => inst_f_sync_type,
            tst_mode => inst_tst_mode, verif_en => inst_verif_en,
            pipe_delay => inst_pipe_delay, reg_count_s => inst_reg_count_s,
            reg_offset_count_s => inst_reg_offset_count_s )
    port map ( clk_s => inst_clk_s, rst_s_n => inst_rst_s_n,
     init_s_n => inst_init_s_n, en_s => inst_en_s,
     count_s => count_s_inst, offset_count_s => offset_count_s_inst,
      clk_d => inst_clk_d, rst_d_n => inst_rst_d_n, init_d_n => inst_init_d_n,
      count_d => count_d_inst, test => inst_test );


end inst;

-- Configuration for use with a VHDL simulator
-- pragma translate_off
library DW03;
configuration DW_gray_sync_inst_cfg_inst of DW_gray_sync_inst is
  for inst
    -- NOTE: If desiring to model missampling, uncomment the following
    -- line.  Doing so, however, will cause inconsequential errors
    -- when analyzing or reading this configuration before synthesis.
    -- for U1 : DW_gray_sync use configuration DW03.DW_gray_sync_cfg_sim_ms;  end for;
  end for; -- inst
end DW_gray_sync_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_gray_sync_inst( inst_clk_s, inst_rst_s_n, inst_init_s_n, inst_en_s,
                  count_s_inst, offset_count_s_inst, inst_clk_d, inst_rst_d_n,
                  inst_init_d_n, count_d_inst, inst_test );

parameter width = 8;
parameter offset = 0;
parameter reg_count_d = 1;
parameter f_sync_type = 2;
parameter tst_mode = 0;
parameter verif_en = 2;
parameter pipe_delay = 0;
parameter reg_count_s = 1;
parameter reg_offset_count_s = 1;


input inst_clk_s;
input inst_rst_s_n;
input inst_init_s_n;
input inst_en_s;
output [width-1 : 0] count_s_inst;
output [width-1 : 0] offset_count_s_inst;

input inst_clk_d;
input inst_rst_d_n;
input inst_init_d_n;
output [width-1 : 0] count_d_inst;

input inst_test;

    // Instance of DW_gray_sync
    DW_gray_sync #(width, offset, reg_count_d, f_sync_type, tst_mode, verif_en,
pipe_delay, reg_count_s, reg_offset_count_s)
         U1 ( .clk_s(inst_clk_s), .rst_s_n(inst_rst_s_n), .init_s_n(inst_init_s_n),
              .en_s(inst_en_s), .count_s(count_s_inst),
.offset_count_s(offset_count_s_inst),
              .clk_d(inst_clk_d), .rst_d_n(inst_rst_d_n), .init_d_n(inst_init_d_n),
              .count_d(count_d_inst), .test(inst_test) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2020	DWBB_201912.5	■ Adjusted the material in "Simulation Methodology" on page 6 ■ Adjusted content and title of "Suppressing Warning Messages During Verilog Simulation" on page 6 and added the DW_SUPPRESS_WARN macro
October 2019	DWBB_201903.5	■ Added the "Disabling Clock Monitor Messages" section
March 2019	DWBB_201903.0	■ Clarified license requirements in Table 1-3 on page 3 ■ Added "Enabling minPower" on page 9 ■ Added this Revision History table and the document links on this page